A Host-Agnostic, Supervised Machine Learning Approach to Automated Overload Detection in Virtual Machine Workloads

Eli M. Dow
IBM Research
Yorktown, New York, USA
emdow@us.ibm.com

Jeanna N. Matthews
Clarkson University
Potsdam, New York, USA
jnm@clarkson.edu

Abstract—**This paper evaluates a mechanism for applying machine learning to identify over-constrained IaaS virtual machines (VMs) in preparation for load balancing or VM migration. Herein, over-constrained VMs are defined as those who are not given sufficient system resources to meet their workload specific objective functions. To validate our approach, a variety of workload-specific benchmarks inspired by the most common Infrastructure-as-a-Service (IaaS) cloud workloads were implemented. These workloads were run while regularly sampling VM resource consumption features exposed by the hypervisor. Datasets were curated into nominal or over-constrained according to per-workload objective functions. These per-VM datasets are used to train machine learning classifiers to determine virtual machine over-constraint rules based on workload one-time log analysis. These rules which are learned when the VM is run in one environment are then transferred with the VM to other environments to determine if they are effective for identifying over-constraint in the new environments as well. Our approach is advantageous because VM creators or software distributors may train a runtime over-constraint classifier that can be applied by future users of the VM image. The VM creators often have unique knowledge of expected VM performance and experience in performance tuning that users of the VM lack. Key contributions of this work include: demonstrating which VM resource consumption metrics (features in machine learning) prove most relevant to learned decision trees, and a discussion of the techniques required to generalize this approach across hosts while limiting required up front training expenditure to a single VM and host. Other contributions include a rigorous explanation of the differences in learned rulesets as a function of feature sampling rates, and an analysis of the differences in learned rulesets as a function of VM workload variation. We demonstrate that 1Hz sampling was sufficient while 1/60 Hz sampling was insufficient in all cases. An analysis of feature correlation matrices and corresponding generated rule sets is provided that demonstrates that individual features comprising rule sets tend to show low cross-correlation (below 0.4) while no individual feature shows high direct correlation with classification. We demonstrate workload-specific error percentages below 2.4% with a mean error across workloads of 1.43% (and strong false negative bias) for a variety of representative cloud workloads tested and ruleset portability across hosts.**

Keywords—virtual machines; cloud-computing; IaaS management; decision trees; support vector machines; binary classification; cloud provisioning; resource allocation.

I. Introduction

There are numerous challenges to solve when implementing highly elastic VM resource allocation schemes. Methods for inferring data about a VM under observation can be divided into in-band and out-of-band approaches. In-band approaches require installing one or more agents that monitor and, when necessary, react to the changing parameters of the system under study. Generally, existing systems require the VM operator to install a software agent in the VM, that can be used to gather information and drive a monitor-analyze-remediate management loop capable of making resource allocation decisions. Out of band monitoring, or agentless monitoring, is simpler to install and administer (centralized on the hypervisor not the VMs) and does not restrict the VM owner by imposing an agent or any other requirements into their VM. The lack of interference or imposed requirements on IaaS cloud VMs cannot be understated, as this cloud type caters to arbitrary OS images.

A. In-Band Monitoring and Agents

An in-band approach works best for the type of cloud computing known as Application-as-a-Service (AaaS), in which a single application is developed, tuned, and sold repeatedly to customers who remotely access the VM application. In this model, installing agents to monitor resource availability and VM system performance is an attractive solution since the provider of the service has full access to the virtual machine. In addition, the VM creator has the requisite domain specific knowledge about the application to effectively instrument systems to detect resource over-constraint. Since a single application is tuned and resold as a service, the cost of developing custom agents on a per-application basis is amortized across all instances sold, making it cost effective and practical due to economies of scale at high volume. AaaS cloud computing environments use agent derived metrics to ensure that some service level agreement (SLA) has been met, and if not, consider the machine to be lacking in resources.

However, monitoring VMs for the purposes of performance management or SLA compliance is a nontrivial task. The clear majority of application performance indicators require application-level knowledge. A common solution for performance instrumentation is a framework that application authors can extend if, and only if, they have access to the source code of the software to be monitored. An example of this type of architecture is the Application Response Measurement (ARM) standard [ARM].

In-band monitoring systems are not always possible, as they often require intrusive instrumentation, ongoing computational burden in the VM (especially with heavy weight agents), and substantial development time. In general, agent-based solutions present a problem in the context of Infrastructure-as-a-Service (IaaS) compute clouds. Also, most agent-based solutions are specific to a given operating system and/or specific user space application. This makes them of limited utility, since they are tightly coupled to the workload they monitor. Consider that an operating system under heavy load could potentially cause an agent running within it to starve for resources. This derivative effect of overloading the system which you are measuring reduces the effectiveness of this type of agent.

In-band agents also suffer from several other drawbacks. They need to be written or customized for every workload (operating system and possibly also application). They increase potential security liabilities and are susceptible to attacks from within the monitored object. This model does not fit with the Infrastructure-as-a-Service model where a *laissez-fare* approach is generally required for the provisioned guest as per typical user agreement.

B. Out-of-Band, Agent-less Monitoring

Virtualization technology makes it possible to encapsulate an entire operating system or application instance within a virtual object that can then easily be monitored and manipulated without any knowledge of the contents or behavior of that object. This can be done out-of-band, using general-purpose hypervisor-based agents that are not affected by the behavior of the VM. Out-of-band agents suffer from reduced visibility into the behavior of the system being monitored and more limited response abilities because they can only treat the monitored machine as a black box. Out-of-band agent techniques would be considered prohibitively expensive in many previous dynamic workload balancing algorithms that deal strictly with short-lived processes (on the order of seconds to complete, such as MPI threads of execution) since management overhead is amortized over the length of the compute job. Relatively fast running compute jobs provide little time to amortize management overhead. However, the long running (indefinite execution) nature of VM processes indicate that the overhead associated with the measurement and detection needed for load balancing may be well worth the cost. In addition, for VM processes in enterprise computing environments, the potential consequences of a bad migration decision may justify using higher-overhead machine learning techniques if they ultimately yield better migration decisions.

C. A Vigilant Approach

Our work is inspired by Vigilant, a system based on out-of-band monitoring in which researchers could determine the health and status of a virtual machine based solely on data observations that were external to the VM yet accessible by the hypervisor context [VIGILANT]. The experimental results from Vigilant show that certain types of problem conditions in VMs could be detected out-of-band with high accuracy while avoiding the pitfalls associated with in-band monitoring. Given that the set of data that can be collected by a hypervisor is inherently limited, it can be quite difficult to discriminate between ideal and non-ideal behaving VMs. Vigilant was the first system to combine the ease of out-of-band data collection with machine learning to improve VM systems management, but while Vigilant focused on detecting a limited set of faults (such as halted on error conditions), our own work goes deeper and provides a generalization of their approach in autonomously detecting VM over-consolidation.

The remainder of this section will explain in more detail the Vigilant approach as well as differentiate our own work from theirs. The system used in Vigilant monitored the hypervisor resource requests and VM utilization. A decision tree classifier machine learning method was used to analyze the readings at run time and detect problems *in situ*.

A key element of the Vigilant system was their choice of decision tree classifier. They selected a decision tree for their work principally because of the training simplicity while also noting that the generated trees are easy to interpret by human observers. Perhaps surprisingly they also found their trained tree to be easy to implement with efficient runtime characteristics. However, the Vigilant authors did note that decision trees may not be ideal. They remark that in terms of classification power, a decision tree is generally considered to be a crude precursor to more modern tools such as support vector machines described in the text “Pattern Classification” by Duda et al [PATCLASS]. They go on to specifically state, that based on their experience, a decision tree is sufficiently powerful for analyzing VM runtime metrics. For further information about decision trees interested readers are referred to Mitchell’s “Machine Learning” [MITCHELL].

Vigilant specifically targeted the detection of extremely high CPU utilization in kernel space. In one early experiment, they deployed several virtual Linux instances under the QEMU emulator [QEMU] with each running a different type of workload (web service, mail service, etc.) started at various times (to vary the overall load on the host system). In these experiments, the authors could classify, using a simple decision tree, the case where the workloads from different machines strain the host machine’s resources, as opposed to the case where only one of the virtual machines was under load. Though they omitted the details of that experiment, they indicated that their approach was applicable in diverse settings. Since QEMU underpins the operation of KVM virtual machines it makes sense that their preliminary results from those investigations should coincide with our own experimental data derived from KVM VMs.

Lastly, the authors of Vigilant warn that the application of machine learning to VMs has been chronically hobbled by the limited availability of labeled data that can be used to train the classifier to detect normal and abnormal operation.

The work described herein varies from Vigilant in scope. This work aims to solve a different, broader problem, wherein a combination of multiple machine learning approaches is used to determine if autonomous, out-of-band, general over-constraint detection of VMs as a trigger for live-migration-based VM redistribution actions is possible.

D. Research Questions and Methodology

Motivated by interest into autonomous over-constraint detection for VMs, this work seeks to answer the following open research questions:

1. Which features or resource metrics are most useful in automatically identifying over-constraint in IaaS cloud computing environments?

2. How do the rule sets of learned classifiers vary across common IaaS VM workload types?

3. How portable are the learned classifiers? Specifically, can they be implemented such that an unmodified classifier can travel with the virtual machine to other hosts while retaining applicability?

4. What are the differences in learned classifiers as a function of sampling frequency? Can one determine a lower bound on practical sampling intervals?

The experimental methodology for determining VM over-constraint by machine learning techniques was performed as described in the remainder of this section. First, a test server was configured with a KVM hypervisor. A virtual machine was then created from scratch and configured with a variety of middleware and workloads to be executed while under observation from the custom black-box VM monitoring software (running within the hypervisor context). Both the hypervisor OS and VM were fully up to date with current Ubuntu releases at the time of this experimentation. The workloads installed on the VM are enumerated in section *A: Workload Overview* while data collection methodology is explained in section *B: Experimental Data Recording*. Each workload was run, while data was collected from within the workload itself on the VM, as well as externally visible data collected at the hypervisor level. The combination of workload-aware instrumentation, and black box data sets allow us to curate data for supervised machine learning, specifically we use the workload-aware instrumentation as the source for ground truth of whether the VM is over-constrained or not and then we learn to classify the VM as over-constrained using only the hypervisor-level data.

A. Workload Overview

When trying to classify a set of benchmarks to use as validation cases for our approach of machine learning over-constraint in virtual machines, it is apparent that the breadth of workloads that are run in public clouds cannot be totally encompassed with a few, relatively specific, workload types. To properly examine the results of machine learning of VM over-constraint, a series of representative workloads were identified. These workloads are surrogates for many of the common types of workloads that are run as cloud instances. They span three major categories: HTTP, SQL, and Video Stream Simulation (File/IO dominated) workloads.

1) HTTP-PHP Workload

Workloads are executed against our HTTP and PHP installation by using the *apachebench* benchmark utility[1]. The individual *apachebench* runs were configured to consider unconstrained VM situations when 90% of page responses were received in less than 3 seconds. When more than 10 percent of the page responses take more than 10 seconds, we consider the HTTP VM server to be over-constrained. This threshold was selected based on the observation that web pages that fail to load almost instantly are considered too slow for production use. Consumers invariably become frustrated with latencies and go elsewhere, especially on mobile devices. Individual page responses taking longer than 5 seconds to return are tallied as errors. Using a wrapper script, the number of concurrent *apachebench* requests is scaled iteratively with 10, 20, 30, 40, 50, 100, 150, 200, 250, 300, 350, 400, 500, up to 1000 concurrent requests. Each iteration is run for 60 second durations before the next iteration proceeds to allow a slow ramp up of workload over time. The workload script logs start and end times of each iteration and aborts early, with a flag and time stamp, when an over-constraint situation occurs.

2) Database Workload

To execute a workload against the database instance, we installed the *mysql-slapd* utility was chosen to act as a workload generator[2]. Overload was defined based on any individual slapd invocation output file with an average query latency exceeding 5 seconds. As was done in the HTTP benchmark script, the workload script logs start and end times of each iteration and aborts early, with a flag and time stamp, when an over-constraint situation occurs.

3) I/O Dominated Workloads (Simulated File Serving)

To simulate file I/O oriented workloads performed in the cloud the *filebench* workload engine was selected and configured to emulate the file access patterns of a streaming video server. This configuration simulated new content creation (uploads) and aging out of older content in such a way as to simulate services like YouTube where videos are added, accessed for some period of major activity eventually being accessed infrequently as interest tapers off. The workload has 2 components, one which creates new content to be served and one serving the content. The workload further distinguishes between actively served videos and those which are no longer active (aged content on disk and no longer in memory). The video writing thread continuously produces new files representing new user uploaded content. The number of threads representing streaming users was selected as the independent variable. The streaming thread count was manipulated over the range of [40-60], incrementing by one additional thread per test until the number of reported I/O operations per second fell below a configured threshold (300) for any given test. Each test ran with a given number of streaming threads for at least 60 seconds before moving on to the next test iteration with more

[1] To test our HTTP-based workload we performed a nominal installation of the Apache web server using the "apt-get install apache2" command. Once a minor change was made to the default configuration of Apache to silence a configuration-file related warning, we proceeded to install PHP to create more realistic web content to serve based on the observation that web pages are not simply static HTML. Once installed and configured, a simple PHP page was created to exercises the PHP runtime.

[2] The freely available example database linked from the MySQL documentation page (dev.mysql.com/doc/index-other.html) was installed to provide a reasonable test data set for the MySQL instance workload. The database is described as "employee data (large dataset, includes data and test/verification suite)." After unzipping and importing the reference data set, it was checked manually by listing the table records on the SQL console. Since one table had almost half a million rows of populated data, the design criteria for a sufficiently large dataset was satisfied.

active file streaming threads. Each of the file serving treads streams content from the active file set. The video replacement rate was set to one video replacement every 10 seconds.

B. Experimental Data Recording

To record the purest data set possible, each VM was exercised in isolation on an otherwise idle KVM host. First, our black-box monitoring software was started to monitor for any active KVM virtual machines without the VM running. We start our out-of-band data collection routines early to capture the full machine boot up process as well as a training workload interval and some idle interval, because the patterns of resource consumption at boot-up are often drastically different than when running an operational workload. Our intuition was that this full spectrum would lead to better trained classifiers with fewer false positives triggered by VM restarts while under management that would otherwise appear to demonstrate atypical workload profiles to an out-of-band monitoring utility. To ensure that this hypervisor-based monitoring component periodically scans a host seeking all KVM process parameters. For each KVM process detected on a given host, the logging component starts a dedicated thread to monitor and collect KVM VM-specific information. Each monitoring thread writes to its own distinct log file that is specific to the VM under observation. Consistency and continuity of log files is maintained by naming the files after the UUID of the VM. This approach allows VMs to be stopped, restarted, and moved to other hosts, while permitting merging of VM specific log files to curate larger VM resource data sets.

After starting the hypervisor based monitoring software, the VM under experimentation was started, thus enabling the monitoring software to collect data throughout the VM boot up process until a quiescent state of activity (a few minutes after boot up completion). Next, an individual workload as described previously (e.g., the HTTP workload or the SQL workload, etc.) was started and run until an over-constraint condition was triggered based on the VM workload performance. The monitoring software continued to run until manually stopped, allowing the capture of quiescent and incidental activity (such as default background processed being triggered by the *cron* daemon etc.), as a further set of data to be considered nominal.

After stopping the VM black-box monitoring software, a run log file, named unambiguously with the UUID of the VM was copied for archival and analysis. Values logged during black-box hypervisor data collection are obtained principally by reading from the */proc* virtual file system from the hypervisor instance. Samples were obtained at 1Hz frequency for each of the features monitored and are enumerated in Table 1 with a comprehensive explanation of feature derivation. The log of feature observations is a simple comma separated value file with columns based on observed features (utilization) and each row corresponding to a single observation instant. Row entries were therefore time coherent and columns are feature coherent (an individual feature time series). Post-processing of the instrumented workload that contain performance satisfaction metrics in conjunction with the black box VM feature log (sampled by the hypervisor) allows the VM feature log to be annotated (or curated) as nominal or over-constrained. This annotated data set forms the basis of our supervised machine learning binary classification scheme.

Table 1: Observed Kernel Features. Features labeled in bold showed observations in our out-of-band observation framework, while italicized feature names indicate no usable observations on our hypervisor platform.

Feature	Data Origin	Description
CPU	/proc/[pid]/stat	Amount of time that this process was scheduled in both kernel and user-blospace time as a percentage of all time elapsed.
UT	/proc/[pid]/stat	Amount of time that this process has been scheduled in user mode, measured in clock ticks. This includes guest time, guest_time (time spent running a virtual CPU, see below), so that applications that are not aware of the guest time field do not lose that time from their calculations. Utime value from /proc/[pid]/stat
ST	/proc/[pid]/stat	Amount of time that this process has been scheduled in kernel mode, measured in clock ticks. Stime value from /proc/[pid]/stat
CUT	/proc/[pid]/stat	Amount of time that this process's waited-for children have been scheduled in user mode, measured in clock ticks. This includes guest time, cguest_time (time spent running a virtual CPU, see below). Cu_time value from /proc/[pid]/stat
CST	/proc/[pid]/stat	Amount of time that this process's waited-for children have been scheduled in kernel mode, measured in clock ticks. Cs_time value from /proc/[pid]/stat
GT	/proc/[pid]/stat	Time spent running a virtual CPU for a guest operating system, measured in clock ticks. Guest_time value from /proc/[pid]/stat
CGT	/proc/[pid]/stat	Guest time (GT) of the process's children, measured in clock ticks. Cguest_time value from /proc/[pid]/stat
DLY	/proc/[pid]/schedstat	Obtained from /proc/pid/schedstat, indicates the time spent waiting on a kernel run queue but not executing.
RCK	/proc/[pid]/io	The number of bytes which this task has caused to be read from storage. This is simply the sum of bytes which this process passed to read() and pread(). It includes things like tty IO and it is unaffected by whether actual physical disk IO was required (the read might have been satisfied from page cache)
WCK	/proc/[pid]/io	The number of bytes which this task has caused, or shall cause to be written to disk. Similar caveats apply here as with RCK.
RBK	/proc/[pid]/io	Attempt to count the number of bytes which this process did cause to be fetched from the storage layer. Done at the submit_bio() level, so it is accurate for block-backed filesystems.
WBK	/proc/[pid]/io	Attempt to count the number of bytes which this process caused to be sent to the storage layer. This is done at page-dirtying time.
RXB	*libvirt*	Received bytes from libvirt virDomainInterfaceStats structure.
RXP	*libvirt*	Received packets from libvirt virDomainInterfaceStats structure
RXE	*libvirt*	Receiver side errors from libvirt virDomainInterfaceStats structure
RXD	*libvirt*	Receiver side dropped packets from libvirt virDomainInterfaceStats structure
TXB	*libvirt*	Transmitted bytes from libvirt virDomainInterfaceStats structure
TXP	*libvirt*	Transmitted packets from libvirt virDomainInterfaceStats structure
TXE	*libvirt*	Transmission side errors from libvirt virDomainInterfaceStats structure
TXD	*libvirt*	Transmission side dropped packets from libvirt virDomainInterfaceStats structure
PF1	/proc/[pid]/stat	The number of minor faults the process has made which have not required loading a memory page from disk. Minflt value from /proc/[pid]/stat
PF2	/proc/[pid]/stat	The number of minor faults that the process's waited-for children have made. Cminflt value from /proc/[pid]/stat
PF3	/proc/[pid]/stat	The number of major faults the process has made which have required loading a memory page from disk. Majflt value from /proc/[pid]/stat
PF4	/proc/[pid]/stat	The number of minor faults that the process's waited-for children have made. Cmajflt value from /proc/[pid]/stat
BIO	/proc/[pid]/stat	Aggregated block I/O delays, measured in clock ticks, expressed as a rate per time. Delayacct_blkio_ticks value from /proc/[pid]/stat

A time series plot of all non-zero time-series for one experiment are shown in Figure 1. Series not shown, were static (zero) observations, and were thus not applicable in our experimental configuration.

II. Experimental Results

This section provides an analysis of our experimental results. First, we present an overview of the multivariate data we collected across workloads and our analysis approach. Second, this section describes the features that were found to initially have predictive power from our set of reasonable guess features listed in Table X. Following that, an investigation into the effect of feature observation variance on learned rules is presented before moving on to quantify rule set accuracy and a brief analysis of surprises encountered during experimentation (related to assumptions about workload-specific predictive features). This section concludes with an analysis of the effects of observation sampling frequency on variation and the portability of learned rule sets.

A. Multivariate Analysis Plots

For each workload run, feature-specific data was analyzed using a series of automated plots and analysis using scalable vector graphics (SVG) for manual inspection. This analysis included time series plots of each feature to observe general trends in the series as well as ensuring each observation was recorded correctly (as evident in Figure 1). Box and whisker plots were created for each feature to characterize the variance of feature observations. Histograms of observed features were created as well as a scatterplot of each feature against the determined classification. The histogram plots of observed values were performed to determine what, if any, type of distribution the feature corresponds to. Many machine learning models assume a Gaussian distribution, and surprisingly none of our observations correspond to a Gaussian distribution. In the scatterplot analysis of each observation, the corresponding determined classification was encoded as a 1 or 0, with 1 meaning over-constrained or under-performing the experimental threshold for successful virtual machine operation response, and 0 meaning nominal or well-performing virtual machine response.

Moving beyond univariate analysis of features, an examination of the multivariate data set (across all features) from each experiment was performed. A representative example Andrews plot for the HTTP workload is shown in Figure 2. A type of signal, shown in teal (darker) distinguishes over-constrained data values against background field of gold (lighter) nominal values. This signal is what our machine learning approach attempts to differentiate from the feature vector observations. Since some signal seems apparent in the plot, we next sought to answer if any individual variable had overwhelming predictive power, or if features showed high predictive correlation obviating the need for machine learning on unnecessary features, or even at all.

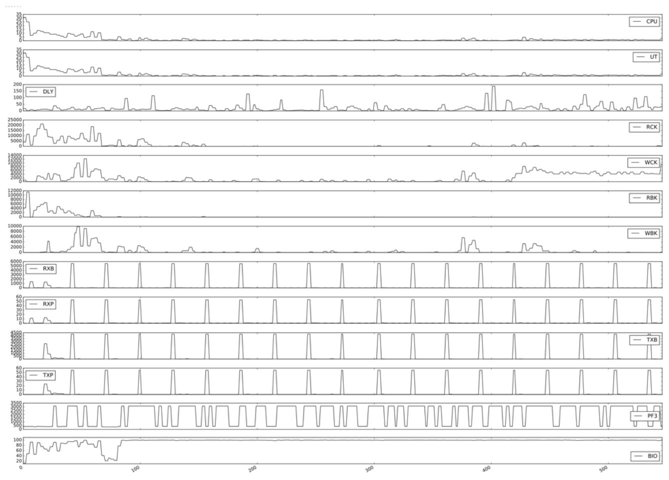

Figure 1: HTTP Experiment Select Time Series. The abscissa shows time in seconds since the beginning of the experiment covering approximately 10 minutes in elapsed time for this experiment. Each vertically stacked subplot shows the time series of observations from a single feature. Only those features with nonzero series are shown. Surprisingly, several seemingly relevant selections for features produced no signal in our measurement framework. Upon investigation, many parameters are not relevant to KVM virtual machines (though would likely be relevant in other hypervisors). Several time series represented in the scale above show high correlation but are in fact distinct time series when viewed at high resolution. The left most 1/6th of the chart corresponds roughly to the boot-up process. All time series values sampled at 1Hz. Series labels correspond to the entries in Table 1.

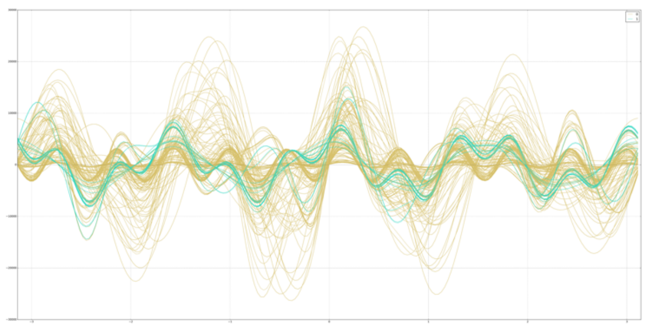

Figure 2: Representative HTTP workload Andrews Plot. Given the high dimensionality of this data, simpler approaches like 2D scatter plots, histograms, and boxplots, are lacking. Teal (dark) lines represent over-constraint, while gold (light) lines represents nominal VM performance. Each attribute of an observational data set row is represented by a point on the line, like a line chart, but the way data is translated into a plot is substantially different. Each column from the data set is normalized independently and smoothed. Andrews Plots are a rolled-down, non-integer version of the Kent-Kiviat radar chart, or a smoothened version of a parallel coordinate plot.

The correlation matrix plots across all feature time series highlight positive and negative correlations in the observations. By focusing on intra-feature correlation, efficiency can be increased by sampling only one element from a highly-correlated feature set at runtime. Example correlation plots are shown in Figure 3, 4, and 5. Since some data are related (received bytes and received packets for instance) some degree of correlation is expected. Our intuition was that an element from a highly-correlated pair would not simultaneously occur in machine learned rulesets. In other words, the machine learning would focus on a single predictive feature from a set of highly correlated features. Also by including the classification in the correlation matrix we could determine if any individual feature was overwhelmingly predictive, implying a simpler univariate approach to over-constraint detection would be sufficient. As shown, no individual feature was highly correlated to the resulting learned binary classification making this problem multivariate, nonlinear, and complex, thus suitable for a supervised machine learning approach.

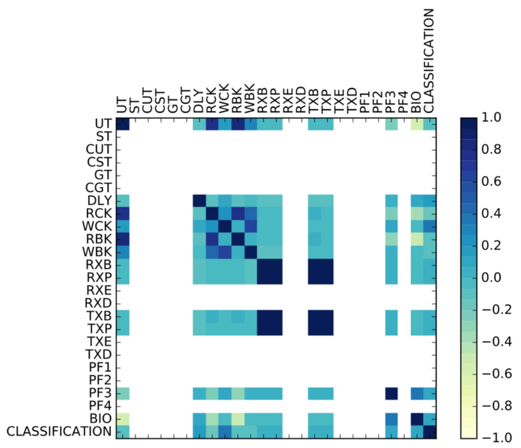

Figure 3: Feature Correlation Matrix HTTP Workload

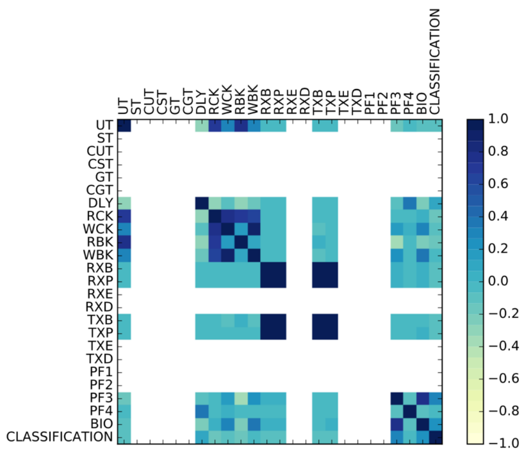

Figure 4: Feature Correlation Matrix SQL Workload

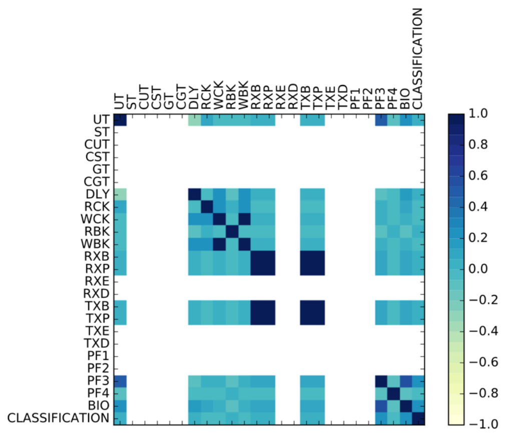

Figure 5: Feature Correlation Matrix Video Workload

B. Initial Features Found to Have Predictive Power

Table 2 summarizes the predictive features from our initial experimentation using 1Hz sampling across workloads.

Kernel Feature	HTTP/PHP Web Server	SQL Database	Video Server (I/O)
CPU	X		X
UT			
ST			
CUT			
CST			
GT			
CGT			
DLY	X		
RCK	X	X	
WCK	X	X	X
RBK			X
WBK	X		
RXB			
RXP			
RXE			
RXD			
TXB			
TXP			
TXE			
TXD			
PF1			
PF2			
PF3			X
PF4			X
BIO	X	X	X

Table 2: Features used in final decision tree classifier rules, separated by workload. Marked values show predictive power in our experimental configuration.

```
Rule 13:
    WCK > 5124.52
    WBK <= 29.33
    -> class 1 [92.6%]

Rule 10:
     CPU > 0.6667
     CPU <= 1.3333
     DLY > 16.59
     WCK > 3103.53
     BIO > 98.33
     -> class 1 [91.2%]

Rule 3:
     WCK > 3103.53
     WCK <= 4104.06
     BIO > 98.33
     -> class 1 [89.1%]

Rule 1:
      WCK <= 3103.53
       -> class 0 [99.6%]

Rule 16:
      RCK > 0.13
      WBK > 29.33
      -> class 0 [99.1%]

Rule 2:
     BIO <= 98.33
     -> class 0 [98.4%]

Rule 6:
     DLY <= 16.59
     WCK <= 5124.52
     -> class 0 [97.8%]
```

Listing 1: HTTP Workload Decision Tree Rule Set

```
Rule 5:
    RCK > 50.77
    WCK > 1.39
    -> class 0 [99.2%]

Rule 1:
     BIO <= 99.67
     -> class 0 [98.8%]

Rule 2:
     RCK <= 50.77
     WCK <= 1.73
     -> class 0 [95.8%]

Rule 3:
      RCK <= 50.77
      WCK > 1.73
      BIO > 99.67
      -> class 1 [88.2%]

Rule 2:
     BIO <= 98.33
     -> class 0 [98.4%]

Rule 4:
     WCK <= 1.39
     BIO > 99.67
     -> class 1 [31.4%]
```

Listing 2: SQL Workload Decision Tree Rule Set

Listing 1, Listing 2, and Listing 3 show resultant rule sets from the initial experimentation. Each enumerated rule within the rules-sets shown is composed of attribute-value arithmetic expressions and the associated classification when the arithmetic expression is true. Each rule expression is followed by a percentage value indicating rule accuracy. Note, that the final rule labels (numbers) are arbitrary and no rules were withheld.

For example, in the rules shown in Listing 1, we can begin to read the rules as follows: Rule 13: if "WCK > 5124.52" and “WBK <= 29.33" then "Over-Constrained”. According to the generated C4.5rules, this rule is accurate 92.6% of the time, or in other words, has a 7.4% margin of error. In all experiments, classification 1 indicates over-constraint, and classification 0 indicates nominal performance. Note that for all rule sets shown, if no individual learned rule was applicable, the default classification assumes nominal or well-behaved VM performance. As shown in Subsection D: Observation Sampling Frequency, the default classification is relevant, as situations with insufficient training data yield default classifications. With trained rule sets, one can invoke the bundled *consultr* rule interpreter application from the C4.5 runtime, or take the approach outlined in [TBDT] to compile custom embodiments of these rules into embeddable, pre-compiled shared objects for use at runtime on a per-VM basis as specified in a virtual machine contract [CONTRACTS].

The remainder of this section attempts to provide the authors intuition on what was expected from the learned rule sets, and what was empirically observed. While not attempting to provide an exhaustive post-hoc analysis of each rule, we do want to convey some surprises we encountered while explaining our intuition. Our intuition is provided to contrast to the reader’s and emphasize the difficulty in forecasting workload performance based on the nuanced interplay of available resources. We will return to major unexpected results from these rules and Table 2 in Section E: Experimental Surprises.

Our intuition of the HTTP workload was that some combination of delay and networking characteristics would dominate the rule set. The results were more nuanced than we had expected. Listing 1 has rules comprised of CPU consumption (CPU), blocked I/O (BIO) and memory writes (in character and bytes as WCK, WBK respectively). Rule 2, the lengthiest rule indicates over-constraint in a narrow range of CPU consumption, when blocked I/O is high, and CPU scheduler delay is high. Simpler rules like Rule 1 and 2 indicate that blocked I/O and memory pressure below a certain threshold indicated an HTTP VM is not likely to be nominal. Similarly, Rule 6 indicates low scheduling delays on the processor with little memory writes are also indicators of nominal behavior.

Our intuition for the SQL workload was that the rules would be generally governed by blocked I/O (BIO) and CPU properties. As shown in Listing 2, our intuition did not entirely match reality. As rules were indeed governed by blocked I/O, but memory characteristics made the remainder of the features of interest. From examination of Rule 2 and 3 we can see a machine learned tipping point for this workload in the WCK property, when in conjunction with high blocked I/O, causes a classification to transition from nominal (Rule 2) to over-constrained (Rule 3).

Having little practical experience with the file streaming workload prior to this research endeavor we had no firm intuition on what to expect. Interestingly the output Rules in Listing 3 indicate that combinations of CPU consumption (CPU), a page fault type (PF3) and blocked I/O (BIO) make up most the rules. Upon reflection, these rules make some intuitive sense since page faults may be related to requesting un-cached content on disk, or requesting content which has aged out. Similarly, processes blocked on I/O seem a relevant indictor of over-constraint for a workload whose sole purpose is to stream I/O content.

```
Rule 9:
     RBK > 52 PF3 > 341.33
     BIO <= 31.67
     -> class 1 [87.1%]

Rule 11:
     PF3 > 368.33
     BIO > 56.33
     BIO <= 78.67
     -> class 1 [79.4%]

Rule 5:
     CPU > 1
     PF3 <= 341.33
     BIO > 56.33
     -> class 1 [50.0%]

Rule 1:
     RBK <= 12
     -> class 0 [99.8%]

Rule 3: PF3 > 320.67
     PF3 <= 341.33
     BIO <= 56.33
     -> class 0 [99.7%]

Rule 6:
     CPU > 16.3333
     PF3 <= 341.33
     -> class 0 [99.6%]

Rule 7:
     WCK > 0
     -> class 0 [99.4%]

Rule 8:
     RBK <= 52
     BIO <= 31.67
     -> class 0 [98.8%]

Rule 10:
     PF3 > 341.33
     PF3 <= 368.33
     BIO > 31.67
     -> class 0 [96.1%]
```

Listing 3: Video Streaming Workload Decision Tree Rule Set

C. Effect of Observation Variance on Learned Rules

Since the data sets collected are inherently multivariate, and many of the features sampled are far more variable than others, experimentation was necessary to explore the relationship between feature variability and machine learned decision tree rules. One method for understanding these workload traces is to visualize the features sampled as a set of coordinates in a high-dimensional data space using 1 axis per variable. This technique is known as Principal Component Analysis (PCA) [PCA][3]. Using this approach, a PCA plot constructs a lower-dimensional projection of the data when viewed from a particularly advantageous viewpoint highlighting variance. In keeping with convention, each workload trace feature was mapped into the range [-1, 1] before plotting the PCA (lowest mode) to determine the features with the largest variance.

In the HTTP experiments PCA results indicate a top tier cluster of influence from TXB, TXP, RXB, RXP, and PF3 features which dwarfed the variance of the remaining features. The conclusion is that our choice of PHP workload variance was highly dynamic with respect to network features and page fault fluctuations. Since the workload was constructed to drive web traffic over intervals with rapidly increasing numbers of clients over TCP/IP connections this makes intuitive sense.

In contrast, SQL experimentation showed the large variance in the PF3, and BIO features. Workload variance was further impacted in a second tier grouping by network performance of both transmission and receipt features, ahead only slightly to a third tier grouping of memory-related features. This makes some intuitive sense as the workload is I/O bound in nature and both page faults and blocked I/O counts can reasonably be expected to rise as the number of clients were increased making SQL requests for dynamic table data until over-constraint conditions were observed in the workload.

The PCA for the file streaming workload demonstrated a cluster of large magnitude vectors forming a top tier with RXP, TXP, TXB, RXB in descending order. These were followed by a second tier with UT, and DLY. A much smaller magnitude was shown for BIO, PF3 followed up with the smallest magnitude vectors for WBK, RBK, PF4, RCK.

In all three of our workload test cases, comparing the PCA feature magnitude and the learned rules in Listing 1, 2 and 3, indicates that feature salience is therefore unrelated to variance for this dataset and problem domain, implying other features of interest might be incorporated into the feature vector of observations with little concern for the variability of the feature impacting the resultant classifier.

D. Rule Set Accuracy

In addition to generating rulesets, the C4.5 runtime also provides measures of rule accuracy against training data reserved for testing at rule generation time. Table 3 provides an analysis of the rule set error and information about the observations used to generate the rule sets. The column labeled “Tested” indicates the number of data rows, or time steps, used in training the classifier with nominal and over constrained data sets.

Workload	Tested	Errors	Error Percentage	False Positive Count	False Negative Count
HTTP	547	6	1.1%	1	5
SQL	260	2	0.8%	1	1
VIDEO	1009	24	2.4%	0	24

Table 3: Decision Tree Accuracy Against Training Data. All values are for a 1Hz sampling regime.

[3] PCA is also known by other names depending on the field of application including but not limited to: the discrete Kosambi-Karhunen–Loève transform in signal processing, the Hotelling transform in multivariate quality control, proper orthogonal decomposition in mechanical engineering, and eigenvalue decomposition in linear algebra.

HTTP Feature	1HZ HTTP INTERNAL NETWORKING	1HZ HTTP EXTERNAL NETWORKING
CPU	X	X
UT		X
ST		
CUT		
CST		
GT		
CGT		
DLY	X	X
RCK	X	X
WCK	X	X
RBK		
WBK	X	X
RXB		X
RXP		
RXE		
RXD		
TXB		
TXP		X
TXE		
TXD		
PF1		
PF2		
PF3		X
PF4		
BIO	X	X
ERROR %	1.1% (6/547)	1.4% (4/287)

Table 4: External Vs. Internal Networking Experiment

Elaborating further on Table 3, the HTTP workload consisted of 547 complete sets of feature observations, each comprising a single row in the VM log. Thus, the experiment took 547 seconds to complete given the 1Hz sampling regime used for these experiments. The "Error" column indicates the sum of false positives and false negatives encountered during rule testing performed by the c4.5 runtime at rule generation time. These values are further broken out into the "False Positive Count" and "False Negative Count" columns, respectively. The "Error Percentage" column indicates the percentage of classifications that would be in error if the rule set was adopted. As reported in Table 3, in general low error percentages are observed while there is a distinct trend in errors toward false negatives.

E. Experimental Surprises

Referring once more to Table 2, the authors encountered a few surprises with the analysis as shown, having expected to see many more of the features outlined in Table 1 to have some predictive value (as evident by writing the data collection routines for what seemed a relevant set of parameters for determining VM health) unencumbered by previous experience with this approach as well as no practical experience working with observations of most of the features sampled. Perhaps most surprisingly was the total lack of network related features that we had expected to see in at least the HTTP workload. To determine the effect of rule set determination on internal vs. externally driven workloads we performed the HTTP workload as driven by an external client using a different bridge network exposing the VM to the outside network, effectively making the guest, the VM under study, and an external workload generation machine act as peers on a flat network. The data collection scheme remained unchanged, save for the experimental networking setup. The results of this analysis are shown in Table 4. As can be seen this change yielded predictive power in a network receipt feature (RXB) as well as a network transmission feature (TXP). The results in Table 4, combined with other time series plots not shown here further prove the correctness of our data collection routines for these features, but demonstrate that it is not obvious which features will show predictive power in learned rulesets. Interestingly, note the indication of PF3 as predictive for this variant while it was not selected in the intra-hypervisor experimental HTTP rule set. While we would intuitively expect some level of transmission and receipt features to be relevant in all HTTP workload variants, the sensitivity of features to hardware platform for the HTTP test was a surprising result. Error rates were small in both the internal and external networking experiments and generally consistent across HTTP variations with slightly more intuitively expected features present in the off-platform client scenario (External Networking). In retrospect, using a decision tree classifier has proven useful to ourselves since a SVM based classifier would work as a black box with no intuition or surprises to guide further investigation.

F. Effects of Observation Sampling Frequency Variation

To determine the effects of feature observation sampling frequency on learned decision tree rules, initial sampling for all VM features exposed to the hypervisor through the *proc* and libvirt interfaces occurred at 1Hz. Intuitively this seemed a reasonably high sampling frequency and was selected, in part, to determine the computational overhead of "high-frequency" sampling. Classifiers were trained as outlined previously, while data sets were post processed (subsampled) by windowing the data into 3, 5, 10, and 60 second data sets to understand the effect of decreased temporal resolution in feature sampling regimes. No averaging was done over the interval of discarded samples to replicate a lower sampling frequency over the same experimental duration. The results of this experimentation are shown in Table 5. There is no simple summary to be provided for the nuanced interplay of sampling regime and predictive features shown in Table 5. However, it is evident that varying the observation sampling frequencies yields a direct and measurable impact on decision tree rules, including accuracy against training tests as well as which features show predictive power. The most commonly predictive feature was found to be blocked IO. Since each of our workloads had an I/O component this is perhaps not entirely surprising. The analysis of rulesets and relating them to intuition and sampling frequency, while interesting, is an aside to the core focus of this work, which was providing reproducible VM over-constraint detection in IaaS VMs so we have intentionally limited discussion here to obvious trends.

In Table 5, sampling was also performed at 1/60 Hz but in all cases, the classifier failed to learn any pattern in over-constraint, and assumed a nominal classification by default. There are two reasonable explanations for this. The first is to assume that the data was simply too sparse, and through sufficient replication of the same experiments, one could build a classifier at the expense of experimental data collection and curation time. A second possible explanation is that the fundamental patterns in the operating system data are on a time scale that is so much faster than once per minute, that transient events indicative of failure/over constraint situations get lost or missed entirely. Another observation is that by decreasing the sampling rate, our machine learning implementation learned a rule set comprising fewer features.

G. Rule Set Portability

To test the portability learned classifiers, a series of portability experiments were performed in which training and classifier learning occurred on a small laptop configuration (4 cores, i5 processor, 2.67 GHz, 3GB RAM). The learned classifier and associated VM were then transferred to a larger server (24 cores, Xeon ES02540 @ 2.5GHz, 94GB RAM) and underwent the VM workload once more (HTTP, SQL, Video Serving). However, in this experiment the log records from the VM under test were used as input to the previously trained classifier rules that were generated on the small host configuration using the same 1Hz sampling interval. In these experiments, the moved classifiers were shown to be portable across hosts.

This is in part due to the design of the input data, that the classifiers are trained on. These training data, or feature values, are not expressed as absolute values or counters as exposed natively in */proc* or libvirt but instead converted to rates per elapsed wall clock time of the nominal sampling frequency. Many of the features sampled are constantly increasing values as they are simply a type of counter (e.g., page fault counters). Using raw counter values would not work in a VM setting because as the VM ages, the counters indefinitely increase. Rules generated from this data would improperly focus on to the absolute value of the counter type features and are therefore not be applicable to an older VM instance whose counters were necessarily higher in magnitude (assuming the same average rate of change in the features over time throughout the VM lifecycle. To make these ever increasing counter relatable across systems is to convert them to the increase per unit time. Similarly, other features that could be expressed as a percentage of absolute physical capacity (CPU related features) were converted into percentages making relation across hosts possible. By using percentages of physical limitation where possible, and rates per unit time everywhere else, we collect data which is abstracted from the physical host and from the age of the VM, and thus rules generated are applicable early in the VMs lifecycle, as well as for long lived VMs with uptimes of months or years).

The results of this experimentation imply classifiers trained on rates which are comparable across host platforms should be host-agnostic within the same general class of host. While we did not have access to exotic hardware, we assume there are limits to this host-agnosticism when drastically different hardware platforms are used (for instance switching to solid state hard drives when training on a host with traditional rotating media). Other hardware platform changes likely to cause impacts to ruleset host-agnosticism include VMs which access file systems over SAN storage vs local media etc. More experimentation across a broad variety of hardware configurations would be required to definitively explore the full nature of classifier portability. Based on the authors' observations of commercial data centers, IaaS host servers are often relatively homogeneous hardware. It has been observed that many hosting data centers purchases full racks of identical servers. It is therefore reasonable to suggest confining VMs and learned classifiers to a cage, rack, or cluster of generally comparable hardware such that sufficiently host-agnostic classifiers could be viable in practice.

Table 5: By varying the feature collection frequency a marked change in the learned rules of the c4.5 classifier is evident. Note that no classifier generated a viable rule set at the 1 Minute sampling interval, and in the case of the SQL workload 1/10 Hz sampling also failed to generate a viable rule set.

VM Feature	HTTP Workload				SQL Workload				Video Server Workload			
Sampling Interval	1 sec	3 sec	5 sec	10 sec	1 sec	3 sec	5 sec	10 sec	1 sec	3 sec	5 sec	10 sec
CPU	X								X		X	
UT												
ST												
CUT												
CST												
GT												
CGT												
DLY	X											
RCK	X				X	X	X					
WCK	X	X	X	X	X				X			
RBK				X					X	X		
WBK	X	X	X									
RXB										X		
RXP						X					X	
RXE												
RXD												
TXB												
TXP			X									X
TXE												
TXD												
PF1												
PF2												
PF3			X						X			
PF4									X			
BIO	X				X	X	X		X	X	X	X
ERROR %	1.1% 6/547	6% 11/183	1.8% 1/55	1.8% 2/110	0.8% 2/260	0% 0/87	1.9% 1/53	3.7%* 1/27	2.41% 24/1009	3.3% 11/337	2.0% 4/292	2.0% 2/101

III. Conclusions and Future Work

The work presented here sought to answer several fundamental questions. The analysis presented indicates low error rates can be achieved in automated detection of over-constrained VM's using binary decision tree classifiers trained on a set of seemingly relevant features exposed to the hypervisor. This technique enables black box (out-of-band, or agentless) classification of VM over-constraint at runtime as a trigger for remediation. Further, this work conclusively demonstrated that the training sampling regime has a large impact on classifier rule generation. Specifically, it was demonstrated that 1Hz sampling was sufficient while 1/60 Hz sampling was insufficient in all cases. Furthermore, the evaluation overhead at 1Hz was negligible using the lightweight *proc* and libvirt based instrumentation that should easily scale to hundreds of VMs without perceptible hypervisor performance impact. This work also demonstrates that when training classifiers in this fashion, it is important to construct a realistic training workload that will behave in the same way as production workloads with respect to network access from outside the hypervisor or within a hypervisor since those training decisions make a marked impact in ruleset determination of predictive features by the classifier runtime. The authors would also like to probe the limits of hardware applicability of generated rule sets, for instance switching a generated rule set to a host with solid state hard drives from a host with traditional rotating media.

Through analysis of feature correlation matrices and corresponding generated rule sets it was demonstrated that features selected within rule sets tend to show low cross correlation (below 0.4) while no individual observed value showed high direct correlation with classification. One possible interpretation is that while there was no “magic bullet” feature that can be used more simply to infer VM health from the hypervisor context, a focused selection on just the right features, surprising as they might be, can be very predictive. Furthermore, these predictive models were successfully trained on a small capacity system and the results moved to a larger capacity production server, and remained functional.

The authors note that the classifier chosen determined predictive feature sets that were relatively small in comparison to the total number of sampled features that were hypothesized to be relevant for this classification effort. Furthermore, generated rule sets were generally short, comprised of between 6 and 9 rules, with each individual rule consisting of at most 5 arithmetic expressions. The authors posit that readability of rules from a binary decision tree classifier would generally be desirable in comparison to a SVM-like classification scheme wherein the resulting learned rule set cannot be crisply articulated to system administrators who, when informally surveyed about the desirability of such an assistive framework (with remedial VM action trigger automatically) generally expressed derision at autonomous system management, however effective, whose policy cannot be explained simply.

Further, our work has demonstrated that VM creators need not ship the workload used to create the classifier with the VM due to abstraction of the learned rule set that operates on black-box VM data without requiring VM instrumentation at deployment time. However, exploring shippable training workloads that could be bundled with a VM for re-training on new classes of hardware may be of interest as new hardware platforms are developed. With the system design used here, the option remains open, but not strictly necessary.

This work is an ongoing investigation into VM management, with the intended application toward triggering agentless, automatic VM rebalancing within an IaaS data center via live-guest migration. However, the approach taken herein is not entirely specific to VMs given that many of the values we use for training and runtime evaluation are simply */proc* entries, and those values which were of libvirt origin may likely have suitable alternative API for general purpose per-process network accounting. If so, the approach demonstrated might apply to non-VM generic workloads encapsulated as individual processes on a host. Thus, extensions to this work may be related in spirit to that of [KUNDU] although that line of inquiry remains beyond the focus of our present research agenda into IaaS cloud management.

IV. References

[1] [ARM] The Open Group. Application Response Measurement — ARM, 4.0 Version 2. http://www.opengroup.org/management/arm/, 2007.

[2] [KILLIAN] T Tom J. Killian created the first implementation of a process file system. It was for Eighth Edition UNIX. See T. J. Killian, "Processes as Files," USENIX Summer Conference Proceedings, Salt Lake City, UT, USA (June 1984).

[3] [LIBSVM] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[4] [QUINLAN] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[5] [TBDT] Eli. M. Dow, and Tim Penderghest. "Transplanting Binary Decision Trees." Journal of Computer Sciences and Applications 3.3 (2015): 61-66.

[6] [PATCLASS] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern Classification. Wiley, 2000.

[7] [MITCHELL] T. M. Mitchell. Machine Learning. McGraw-Hill, 1997.

[8] [QEMU] F. Bellard. Qemu, a fast and portable dynamic translator. In USENIX 2005 Annual Technical Conference, FREENIX Track, pages 41–46, 2005.

[9] [KVM] - A. Kivity. kvm: the Linux Virtual Machine Monitor. In OLS ’07: The 2007 Ottawa Linux Symposium, pages 225–230, July 2007.

[10] [PHP] - PHP website accessed January 2017 at http://www.php.net/.

[11] [MYSQL] – MySQL Database Project. https://www.mysql.com/

[12] [APACHE] – Apache HTTP Server Project. http://httpd.apache.org/

[13] [AB] Apache HTTP Server Benchmarking Tool User Manual- http://httpd.apache.org/docs/2.4/en/programs/ab.html

[14] [ABF] - Using Apache Bench for Simple Load Testing February 05, 2009. http://www.petefreitag.com/item/689.cfm

[15] [IOZONE] Martin, Ben (2008-07-03). "IOzone for filesystem performance benchmarking". Linux.com. Retrieved 2016-2-06:

https://www.linux.com/news/iozone-filesystem-performance-benchmarking

[16] [IPERF] iperf3 utility home page: http://software.es.net/iperf/

[17] [PCA] Pearson, K. (1901). "On Lines and Planes of Closest Fit to Systems of Points in Space" (PDF). Philosophical Magazine. 2 (11): 559–572. doi:10.1080/14786440109462720.

[18] [KUNDU] Kundu, Sajib, et al. "Modeling virtualized applications using machine learning techniques." *ACM SIGPLAN Notices*. Vol. 47. No. 7. ACM, 2012.

[19] [CONTRACTS] J. Matthews, T. Garfinkel, C. Hoff, and J. Wheeler, Virtual machine contracts for datacenter and cloud computing environments, in Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds, 2009, pp. 25-30

[20] [AP] García-Osorio, César; Fyfe, Colin (2005). "Visualization of High-Dimensional Data via Orthogonal Curves" (PDF). Journal of Universal Computer Science. 11 (11): 1806–1819.